Attack Pattern Generation

Sean Barnum, Cigital, Inc. [vita[3]]
Amit Sethi, Cigital, Inc. [vita[4]]

2006-11-07 L3 / L, M[5]

This article is the second in a coherent series introducing the concept, generation, and usage of attack patterns as a valuable knowledge tool in the design, development, and deployment of secure software. It is recommended that the reader review the Attack Patterns Introduction article to fully understand the context of the material presented.

The Introduction to Attack Patterns[6] article presented introductory and contextual information on attack patterns. This article describes a typical process for how attack patterns are actually generated. The intended audience for this article includes mainly security researchers and experienced security practitioners who are interested in discovering and documenting new attack patterns. It is, however, also valuable for the broader audience in that it gives a much deeper understanding of the source and meaning of attack patterns.

Purpose

As attackers become more sophisticated, they will discover new ways of exploiting software. In addition, new software and development environments will introduce new types of vulnerabilities that presently may be unknown. To ensure that the software development community continues to implement effective countermeasures against the latest known attacks, it is important to analyze the latest exploits to see whether they represent any new types of attacks. Only after the attacks are characterized can effective countermeasures against those types of attacks be designed and implemented.

In addition, analyzing the latest exploits and generating new attack patterns is an essential prerequisite to the creation of effective security policies. Policy developers should be aware of all known attacks that a system may face before they attempt to develop relevant and complete security policies.

While attack patterns fundamentally represent common approaches to exploits, they do not necessarily need to be generated only from actual exploits discovered in the wild. In cases where organizations are performing security-related research, they may discover a new way to attack a system. This knowledge can be used internally by the organization (or provided to vendors) to mitigate the issues before attackers discover them. The example presented at the end of this article was actually discovered in this manner.

Inputs

The process of attack pattern generation begins when a new exploit is discovered that does not match one of the known attack patterns. The inputs to the process include the actual exploits (if available), any existing vendor patches for the exploits, forensic information, and the existing knowledge base of attack patterns. The exploits may be discovered in the wild, or they may be generated by security researchers in an attempt to discover exploits before attackers do.

Analysis

The exploit analysis process typically consists of the following steps:

1. Analyze the exploit through reverse engineering, forensic analysis, and analyzing any available patches by vendors of the target software. This step is not specific to attack pattern analysis and is generally

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)
4. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/601-BSI.html (Sethi, Amit)
6. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/585-BSI.html (Introduction to Attack Patterns)

performed to understand exploits and develop countermeasures such as antivirus definitions and spyware removal tools. Once the inner workings of the exploit are revealed, actual attack pattern analysis can begin.

2. Determine whether the exploit is an instantiation of any existing attack patterns. This is often not a clear and unambiguous decision. A careful analysis and comparison must be performed. In most cases where the exploit is discovered in the wild, it will be an existing attack pattern and the analysis will stop here. Otherwise, a new attack pattern will have been discovered, which should be analyzed and documented as described below.
3. Determine the functionality in the software that contained the vulnerability. The functionality could be a file parser, format converter, cookie handler, or anything else. Determine whether the exploit attacks a vulnerability or weakness in the particular functionality or whether the same issue could exist in the software even if the targeted functional component were removed. If the exploit targets specific functionality, then attempt to generalize the attack. For example, if an attack targets an MP3 player, then could similar attacks also be used against WMV players, JPEG viewers, ZIP file extractors, etc? The vulnerability could lie in *any* binary file processor, or it could exist *only* in MP3 playing software.
4. Determine how the software vulnerability was exploited. Examples include providing a maliciously crafted file to the software, leveraging a race condition, providing separator characters in the input, or bypassing client-side input filtering. This step helps to identify how the targeted functionality determined in the previous step was exploited.
5. Determine what skill level or knowledge the attacker would need to execute such an attack. Note that there may be different skill levels and knowledge required to generate certain results. For example, exploiting a buffer overflow to crash a system may require very little skill, but actually executing malicious code on the target host to gain control of it may require a highly skilled attacker.
6. Determine the resources required to execute the attack. Does the attack simply require an attacker manually entering commands at a terminal, or does it involve compromising thousands of hosts before using them to attack the main target? Would execution of the attack require a well-funded organization's support? It is important to determine the resources required to execute an attack, as it helps determine the likelihood of an attack and helps to prioritize mitigations during actual software development.
7. Determine the motivation of the attacker that generates this type of exploit. Why would an attacker choose this type of attack in particular? Given a choice between various technical (e.g., executing a buffer overflow) and non-technical (e.g., social engineering) means of achieving a goal, attackers tend to select the easiest ones. Keeping that in mind, determine what makes the particular attack attractive. What does the attacker have to begin with, and what does the attacker want to accomplish? This discussion should be mostly technical, because business consequences will obviously depend on the particular software and a deployed environment. The consequences may include execution of arbitrary code on target host, denial of service, obtaining privileged access to target host, etc.

Evaluation

Once a new attack pattern is generated, it should be evaluated to ensure that it models the applicable exploits well. It is essential that the evaluation be performed by a different person than the one who generated the attack pattern. Otherwise, the attack pattern author could potentially gloss over issues due to implicit assumptions made during the analysis. The first step is to ensure that a pre-existing attack pattern does not model the exploits. If a pre-existing attack pattern that models the exploit is found, determine why the new attack pattern was created and whether it would be more beneficial to amend the existing attack pattern than to create a new one.

Assuming that the attack pattern is new, ensure that the exploits from which it was generated are actually instantiations of the attack pattern. If not, determine what modifications to the attack pattern are required to correct the problem. If this is done, then examine the unmodified attack pattern as well to determine whether it is a valid attack pattern that may be applicable in other scenarios.

The next step is to ensure that the attack pattern is not overly generic. Questions that may help to determine this include:

- Does the attack pattern describe what part of the software is attacked?
- Does the attack pattern describe how malicious input is provided to the target software?
- Will knowledge of the attack pattern help a software designer or developer avoid the problem?

If the answer to any of the above questions is "no," then the attack pattern is likely too generic. For instance, an overly generic attack pattern may state "providing an unexpectedly large input to the application causes it to crash." This "attack pattern" does not describe what part of the software is attacked or how malicious input is provided to the target software. Developers do not learn what input they need to validate (e.g., input from text fields in a Windows GUI application, input from a web form, input from a binary resource file). It provides no indication as to what particular source of input is untrusted, and hence must be validated. A potential attacker also would be unable to use this attack pattern because it tells them absolutely nothing about how malicious input could be provided to the application. This may be considered positive, but the fact is that many attackers are likely already familiar with the attack, along with specific exploit instances. At the same time, if the attack pattern is completely useless to an unskilled attacker, it likely does not represent a valuable capture of the attacker's perspective. More details can be provided to make the attack pattern helpful to software developers, while keeping particular attack details private.

The next step is to ensure that the attack pattern is not overly specific. Questions that may help to determine this include:

- Can the attack pattern be used to find previously undiscovered vulnerabilities in software?
- Can the attack pattern be used by relatively unskilled attackers to exploit software?

If the answer to the first question is "no," then the attack pattern is likely too specific. If the answer to the second question is "yes," then either the attack is extremely simple (such as a command line delimiter attack) or the attack pattern is too specific and detailed. If the attack pattern is found to be too specific or detailed, this problem should be mitigated. An overly specific attack pattern is likely to benefit only attackers and provide little ongoing value to developers.

The accessibility of the attack pattern also should be evaluated; one of the target audiences is software designers and developers that may have little or no training in security issues. It would be easy for security researchers to develop attack patterns that are completely inaccessible to designers and developers. It is important to keep the goal of creating attack patterns in mind: attack patterns are designed to help software designers and developers with little security experience to understand security issues so that they can develop secure software.

Outputs

The typical schematic structure and content for an attack pattern that comes out of this process is described in the Introduction to Attack Patterns[12] article. In addition to the information described there, the attack pattern should clearly indicate the author(s) and reviewer(s) of the attack pattern. Depending on the environment in which the attack pattern was developed, it should be published either to a private company repository or to a public attack patterns repository. This will ensure that the attack pattern will be easily accessible and will not get lost.

Example

We will illustrate the attack pattern generation process using an example. The example used here was chosen for its simplicity of explanation and understanding and was, in reality, discovered through other means.

Suppose that there have recently been many reports of *gzip*-compressed files that have been causing most virus scanners to crash. In addition, receiving *gzip*-compressed HTML data is causing popular web browsers

12. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/585-BSI.html (Introduction to Attack Patterns)

to crash. A security researcher, Alice, has been given the task of investigating the issues and determining whether they are related.

Alice is already familiar with existing attack patterns that are public knowledge (assume that this particular attack is not public knowledge). She begins by obtaining some problematic *gzip*-compressed files. She tries to decompress one and notes that the decompression utility takes an unusually long time. She obtains a file listing of the directory in which the file is being decompressed and notes that the file being output by the decompression utility is currently over 1 gigabyte in size. Realizing that this is unusual, she stops the decompression utility and opens the partially decompressed file in a hex editor. The decompressed file seems only to contain the letter ”A,” repeated over and over. Knowing how run-length encoding works, Alice realizes that the decompressed file must simply contain the same byte repeated billions of times, so that the compression is extremely efficient. The compressed file more or less need only contain the letter ”A,” along with the number of times it is repeated in the original file. Thus, the compressed file can be extremely small (several kilobytes to several megabytes), but the decompressed file can be several hundred gigabytes in size. When a virus scanner attempts to scan the contents of the compressed file, it must decompress it in memory first. Most computers do not have several hundred gigabytes of memory and eventually run out of memory and crash. Alice discovers the same problem at malicious websites that send a small amount of *gzip*-encoded HTML data to the clients, causing the browsers that attempt to decompress and display the data to crash.

Now that Alice has discovered the cause of the attack and knows that the attack pattern does not currently exist, she decides to generate an attack pattern that describes the attack. She determines that the functionality under attack is decompression of *gzip*-encoded data. However, she also realizes that *gzip* is not the only type of encoding that may be susceptible to such attacks. Other encoding formats, including *ZIP*, *bzip2*, *PNG*, and *GIF*, also use run-length encoding to compress data and could be vulnerable to such attacks. In fact, any type of compression where the efficiency of compression is not bounded by a reasonable value may be vulnerable to such attacks. Thus, Alice determines that any software that performs decompression may be vulnerable to such attacks.

Next, Alice must determine how the vulnerability is exploited. She determines that this problem could occur any time a target host attempts to decompress malicious data. All an attacker must do is activate software on the target host to attempt to decompress the data in memory. Software that may do this includes virus scanners, image viewers, and web browsers. Thus, the attacker must somehow place the malicious data on the target host. In most cases, a utility such as a virus scanner would automatically do the rest. In fact, this can be used to cause a fairly large-scale denial-of-service attack. The malicious file can be e-mailed to an address in a target corporation, and the virus scanner on the e-mail server will cause the server to crash. Even if backup servers exist, they will also crash when attempting to pick up where the other server left off (i.e., when they attempt to scan the malicious file). This will effectively cause a denial of service and will halt all e-mail communication within that organization.[14]

Now, Alice must determine the skill level of the attacker that may be able to execute such an attack. She notes that crafting a malicious file is not an simple task, as the attacker cannot just create a file that is several hundred gigabytes in size and then compress it (due to memory limitations of the attacker’s computer). The attacker must know details of the compressed file format so that he/she can craft a valid but malicious compressed file without having to create the original decompressed file. However, if such a malicious file is ever obtained by an unskilled attacker, he/she can easily leverage it to attack other systems. Thus, creating a malicious file requires a skilled attacker, but even an unskilled attacker can use the file to execute an attack.

Alice next must determine the resources required to execute the attack. This particular attack requires minimal resources--simply that the attacker be able to send data to the target, which could be done via e-mail, HTTP, FTP, etc.

Now that Alice knows the attack details, she needs to determine why an attacker would execute such an attack. Identifying the motivation helps to determine the relevance of this attack pattern to future situations. This attack is focused on achieving a denial of service. An attacker wanting to disrupt all e-mail

14. This issue is now public knowledge and has been mitigated in all popular antivirus software. Public release of this information in this paper no longer poses a significant threat.

communication in an organization, either for simple mischief or to cause losses, may try to execute this attack. The attack is extremely simple, and a single e-mail can halt all e-mail communication within an organization regardless of the number of backup mail servers. The attacker may also want to deny all users access to a message board by posting a malicious picture on the message board that causes users' browsers to crash when they attempt to view it. A professional attacker wishing to monetize such an attack could even leverage this attack in combination with other business actions, such as bringing down a T-bill auction at an opportune time to manipulate monetary results. There are several such attacks where an attacker can deny the target access to some resource for a limited period of time.

The following describes the attack pattern:

1. **Pattern name and classification**: Denial of Service – Decompression Bomb
   - **Attack Prerequisites**: The application must decompress compressed data. For the attack to be maximally effective, the decompression should happen automatically without any user intervention.
   - **Description**: The attacker generates a small amount of compressed data that will decompress to an extremely large amount of data. The compressed data may only be a few kilobytes in size, whereas the decompressed data may be several hundred gigabytes in size. The target is running software that automatically attempts to decompress the data in memory to analyze it (such as with antivirus software) or to display it (such as with web browsers). When the target software attempts to decompress the malicious data in memory, it runs out of memory and causes the target software and/or target host to crash.
   - **Related Vulnerabilities or Weaknesses**: CWE-Data Amplification[15], CVE-2005-1260[16]
   - **Method of Attack**: By maliciously crafting compressed data and sending it to the target over any protocol (e.g., e-mail, HTTP, FTP).
   - **Attack Motivation-Consequences**: The attacker wants to deny the target access to certain resources.
   - **Attacker Skill or Knowledge Required**: Creating the exploit requires a considerable amount of skill. However, once such a file is available, an unskilled attacker can find vulnerable software and attack it.
   - **Resources Required**: No special or extensive resources are required for this attack.
   - **Solutions and Mitigations:** Restrict the size of output files when decompressing to a reasonable value. Especially, handle decompression of files with a large compression ratio with care. Builders of decompressors could specify a maximum size for decompressed content and then cease decompression and throw an exception if this limit is ever reached.
   - **Context Description:** Any application that performs decompression of compressed data in any format (e.g., image, archive, sound, gzip-ed HTML)
   - **References:** Decompression bomb vulnerabilities[17]

Further Reading

To learn more about the concept of attack patterns and how they can benefit you, it is recommended that you read the remaining articles in this series. They provide a clear understanding of how you can effectively apply the use of attack patterns to improve security enablement[18] of the software development lifecycle. The series also includes a detailed glossary[19] of terms, a comprehensive references[20] listing, and recommendations for further exploration[21] of the attack pattern concept.

18. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/588-BSI.html (Attack Pattern Usage)
19. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/590-BSI.html (Attack Pattern Glossary)
20. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/587-BSI.html (Attack Pattern References)
21. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/589-BSI.html (Further Information on Attack Patterns)

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com